UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/676,889	09/30/2003	Shih-Wei Liao	42P16806	8089

45209 7590 07/14/2010

INTEL/BSTZ
BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
1279 OAKMEAD PARKWAY
SUNNYVALE, CA 94085-4040

EXAMINER	
MITCHELL, JASON D	

ART UNIT	PAPER NUMBER
2193	

MAIL DATE	DELIVERY MODE
07/14/2010	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/676,889	LIAO ET AL.	
	Examiner	Art Unit	
	Jason D. Mitchell	2193	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *30 April 2010*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-6 and 8-29* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-6, 8-29* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

This action is in response to an amendment filed on 4/30/10.

Claims 1-6 and 8-29 are pending in this application.

Response to Arguments

Claims 8-9

Claim 8 presents new limitations (i.e. a synchronization period based on a distance between the one or more helper threads and the main thread), not previously considered and addressed with new grounds of rejection in this action.

Claims 1-2 and 15-16

On 12 the applicant asserts the rejection of claims 1 and 15 is improper for the reasons given, presumably, in regard to claim 8. Those arguments (presented on pp. 11-12) assert the Luk references fail to teach the claimed limitations because they teach "three new instructions to allow a main thread to spawn and terminate a pre-execution thread" and the claims require synchronization codes based on a period and that the period is based on a distance between the helper threads and the main thread. For example see the last partial par. on pg. 11, through the first full par. on pg. 12:

> Rather, Luk teaches extending an instruction set with three new instructions to allow a main thread to spawn and terminate a pre-execution thread (Luk, sec. 3.1, page 44). Luk specifically states that only a main thread, i.e. not a pre-execution thread, is allowed to spawn and terminate a pre-execution thread (Luk, sec 3.1, page 44). Luk also describes hardware based heuristics for a hardware to terminate a pre-execution thread. (Luk, sec 3.5, page 46). Thus, Luk's pre-execution threads may be controlled by a main thread or a hardware mechanism. However, Luk is completely silent about generating software codes including synchronization codes

> for a helper thread to synchronize with a main thread during execution. Further, Luk I teach that the pre-execution is to stop (PreExecute_Stop command) at the end of each loop, to prevent runaway threads. Luk I also teach an error avoidance mechanism where the thread is terminated if the next PC is out of range. Neither of these techniques are synchronization methods.
>
> Instead, Luk I teach mechanisms to avoid runaway threads and PC overflow, but merely stopping the threads. In contrast, Applicants' claimed invention requires that the synchronization codes are based on a period so that the one or more threads may synchronize with each other, and the period is based on a distance between the helper threads and the main thread. Luk I do not teach or suggest periods being used to aid synchronization.

The examiner respectfully disagrees. First it is noted that the limitation presented in claim 1 does not make reference to either the period based on a distance between the helper threads and the main thread or the plurality of helper threads recited in claim 8. Instead the limitation in question recites:

> … selecting the code region further comprises determining a synchronization period for the helper thread to synchronize the main thread and the helper thread, the helper thread performing its tasks within the synchronization period; …

As the applicant has indicated Luk's "PreExecute_Start" and "PreExecute_Stop" allow a main thread to spawn and terminate (i.e. synchronize) a pre-execution thread. Further, as also indicated by the applicant Luk I teaches that the "PreExecute_Stop" command causes a pre-execution thread to stop executing at the end of each loop. This defines a period (i.e. a loop iteration) in which the pre-execution thread performs its tasks. Accordingly it is not clear why claim 1 should be considered to distinguish over the Luk references.

Claims 3-4

The applicant asserts claims 3-4 are patentable for the reasons discussed (and found unpersuasive) regarding claim 1.

Claims 5-7, 17-19, 22-25 and 28-29

Claim 5:

On pg. 13 the applicant asserts the Luk references and Annavaram do not teach building dependent graphs from the source code at compile time. Specifically see pg. 13, last full par.:

> Regarding Claim 5, Annavaram seem to teach generating dependence graphs as runtime (see at least the Abstract, line 15). Dependencies are determined based on instructions in the instruction fetch queue (see, at least, page 52, second full paragraph). Further, it seems that the dependence graph generator of Annavaram is a hardware construct added to the system (See Figure 1, page 53). In contrast, Applicants' claimed invention builds dependent graphs from the source code, not at runtime. Analyzing source code by a compiler is not the same as hardware to generate graphs during runtime, of instructions in the fetch queue. Thus, the cited art fails to teach or suggest all of the limitations of the recited claims.

The examiner respectfully disagrees. The applicant is arguing against the Annavaram reference individually when the limitation was rejected over the combined teachings of Annavaram and the Luk references. Annavaram is not relied upon for a teaching of analyzing source code at compile time. This aspect of the limitation is disclosed by Luk.

Luk I discloses a compiler analyzing source code to identify loads which may experience cache misses (pg. 44, col. 2 3rd full par. “the compiler … needs to perform … locality analysis phase which determines which references are likely to cause cache misses”) but does not explicitly disclose building the claimed “dependent graph” or

performing the claimed "slicing operations". Annavaram teaches building the claimed "dependent graph" and performing the claimed "slicing operations" (e.g. pg. 1 Abstract "efficiently generates the required dependence graphs"; pg. 9, par. bridging col. 1 & 2 "slices are preexecuted … for early generation of load addresses"). Accordingly, all aspects of the limitation are taught by the combination of Luk I and Annavaram, and the applicants' arguments that Annavaram does not teach analyzing source code at compile time are not persuasive.

Claims 17, 23 and 29:

In the par. bridging pp. 13 and 14 the applicant indicates claims 17, 23 and 29 have been amended to recite:

> … wherein the one or more helper threads are synchronized with the main thread based on a synchronization period determined for the one or more helper threads o that the one or more helper threads perform the one or more computations during the synchronization period …

And argues that claims 17, 23 and 29, and there dependents are patentable for the reasons discussed, and found unpersuasive, above.

In the first two full paragraphs on pg. 14, the applicant argues that Luk does not disclose determining a synchronization period for the helper thread because Luk discloses terminating a pre-execution thread if its next PC is out of an acceptable range imposed by an operating system. Specifically the 2nd full par. on pg. 14 states:

> Instead, Luk terminates a pre-execution thread if its next PC is out of an acceptable range imposed by an operating system to preserve correctness (Luk, pg. 46, col. 1, 2nd par). Here, Luk merely exhibits the common-sense approach that a thread

> should be terminated if the PC for the next instruction falls outside the acceptable range of executable addresses of the operating system. Whether the PC of a thread falls outside the acceptable address range imposed by an operating system has nothing to do with whether the thread synchronizes with a main thread. Thus, Luk does terminate a thread when the PC of the thread is out of an acceptable range. It is respectfully submitted that one with ordinary skill in the art would not recognize determining a synchronization period for a helper thread to synchronize a main thread with the helper thread within the synchronization period based on Luk's teaching.

The examiner respectfully disagrees. First it is noted that the rejections of claims 17, 23 and 29 do not rely on a citation to Luk's pg. 46, col. 1, 2nd par. (nor does pg. 10 of the previous office action). Accordingly it is believed that this argument is directed to the rejection of claim 20 on pg. 20 where the citation was made.

Claim 20 reads in relevant part:

> … determining a time period for each of the helper threads in the run queue, each of the helper threads being terminated from the run queue when the respective time period expires thereby releasing resources back to the main thread.

Note that the 'time period' is not the same as, nor does the limitation make reference to a 'synchronization period'. Accordingly the applicant is arguing limitation not presented in the claim and those arguments are unpersuasive. Further note that the limitation in question is addressed with a combination of the Luk I and Klemm references. Specifically, Luk 1 discloses "there is a default limit on the number of instructions that a pre-execution thread can execute" (pg. 46, col. 1, 2nd par.) Although not a time period, this describes a system enforced maximum execution length (or broadly a 'period') which, if reached, will cause the thread to be terminated "thereby releasing resources back to the main thread" as claimed (see e.g. pg. 45, col. 1, 2nd par. "Once this limit is reached, the thread will be terminated any way").

Klemm teaches a similar system enforced maximum execution length except that in Klemm the length is defined in units of time and not instructions (col. 5, lines 57-58 “thread execution time exceeds user-specified threshold”). Accordingly the combination of Luk I and Klemm teaches imposing a maximum time period on each of the helper threads as claimed.

Claims 10-11

The applicant asserts claims 10-11 are patentable for the reasons discussed (and found unpersuasive) regarding claims 3-4.

Claims 12-14

The applicant asserts claims 12-14 are patentable for the reasons discussed (and found unpersuasive) regarding claims 5-7.

Claims 20-21 and 26-27

On 15, the applicant asserts the combination of the Luk references, Annavaram and Klemm do not teach determining a time period for each of the helper threads because Klemm is directed to non-helper threads. For example see pg. 15, last par.:

> The Examiner asserts that Klemm teach determining a time period for a thread (col. 5, lines 57-58 "thread execution time exceeds user-specified threshold"). Klemm seem to teach a user-specified threshold for a thread to run to avoid a thread hang. However, the threads Taught by Klemm are for a multi-threaded Java application and are not related to pre-fetching memory accesses, or helping a main thread. Thus, while the art may appear on its face to be analogous, the implementing threads according to Klemm will not result in the same type of result as the recited helper threads. Further, Klemm teaches terminating a thread when it has been

> determined to be hung (based on a pre-determined threshold), but Applicants terminate helper threads when it is predicted that the helper thread is not needed, in order to release resources that may be used by the main thread. Moreover, Claims 20-21 and 26-27 are dependent on allowable base claims, as discussed above.

The examiner respectfully disagrees. Again the applicant is arguing against the references individually where the rejection was based on a combination of references. Specifically, the combination presented in the rejection does not implement Klemm's threads. Instead it implements Klemm's trigger for thread termination. Further, those of ordinary skill in the art would have understood that the thread termination trigger taught by Klemm is applicable to all threads and not just threads for a multi-threaded Java application (i.e. any thread can 'hang' regardless of its intended purpose). Finally, it is noted that the claims are not directed to predicting if the helper thread is needed or not, only to terminating the thread after a timer has expired.

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claims 1-2 and 15-16 are rejected under 35 U.S.C. 103(a) as being unpatentable over "Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors" by Luk (Luk I) in view of US 2003/0084433 to Luk et al. (Luk II).

Regarding Claim 1: Luk I discloses a method, comprising:

analyzing source codes of a main thread having one or more delinquent loads, the one or more delinquent loads representing loads which likely suffer cache misses during an execution of the main thread (pg. 44, col. 2 3rd full par. “locality analysis phase which determines which references are likely to cause cache misses”; also see Appendix Phase I, Step 1; pg. 45, "data address generation and the surrounding control flow"), the source codes including one or more code regions, each code region corresponding to a sequence of instructions representing an iteration loop in the source codes, the one or more code regions sharing at least one instruction in the source codes (pg. 42, col. 1, 2nd full par. “when we are still working on the current list … nodes on the next list can be loaded speculatively by pre-executing the next iteration of the for-loop”; pg. 42 Fig. 2(b), “PreExecute_Start(END_FOR); … PreExecute_Stop()” and the corresponding region in Fig. 2(a));

selecting a code region from the one or more code regions for one or more helper threads with respect to the main thread based on the analysis (pg. 44, col. 2 3rd full par. “locality analysis phase which determines which references … could benefit from pre-execution”; also see the Appendix Phase I, Step 2 “control-flow and call-graph analysis”), selecting the code region further comprises determining a synchronization period for the helper thread to synchronize the main thread and the helper thread, the helper thread performing its tasks within the synchronization period (pg. 42, col. 1, 3rd

full par. “PreExecute_Stop(), which will terminate the pre-execution and free up T for future use”); and

generating code for the one or more helper threads, the one or more helper threads being speculatively executed in parallel with the main thread to perform one or more tasks for the selected code region of the main thread (pg. 44, col. 2, 3rd full par. “performs all necessary code transformations”; also see the Appendix Phase II).

Luk I does not explicitly disclose estimating a communication cost between the main thread and each code region.

Luk II teaches estimating a communication cost between the main thread and each code region (par. [0029] “examining the benefit of load prefetching versus the cost in including the prefetches … The costs generally include the overhead associated with prefetch instructions the additional memory bandwidth consumed").

It would have been obvious to one of ordinary skill in the art at the time the invention was made to estimate communication costs between the main thread and each code region as taught by Luk II (par. [0029] “examining the benefit of load prefetching versus the cost in including the prefetches”) when selecting Luk I's code regions (pg. 44, col. 2 3rd full par. “locality analysis phase which determines which references … could benefit from pre-execution”). Those of ordinary skill in the art would have been motivated to do so to ensure that prefetching would actually provide a benefit (Luk I par. Luk II par.

[0030] "favors prefetching a load if the data can be prefetched and then loaded in fewer clock cycles than it would take to load the data from memory").

Regarding Claim 2: The rejection of claim 1 is incorporated; further Luk I discloses identifying the region comprises:

generating one or more profiles for cache misses of the selected code region (pg. 43, the par. bridging the cols. "based on profiling information"; also see the Appendix, Phase I, Step 1 "This step can be accomplished through some low-overhead profiling tools"); and

analyzing the one or more profiles to identify one or more candidates for thread-based prefetch operations (pg. 43, the par. bridging the cols. "the compiler usually needs to heuristically decide how to prefetch … based on profiling information").

Regarding Claims 15-16: Claims 15-16 recite a system for performing the method of claim 1 and are addressed similarly.

Additionally claim 16 recites and Luk I discloses the process is executed by a compiler during a compilation of an application (pg. 44, col. 2, 3rd full par. "the compiler … is responsible for inserting pre-execution").

Claims 3-4 are rejected under 35 U.S.C. 103(a) as being unpatentable over "Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors" by Luk (Luk I) in view of US

2003/0084433 to Luk et al. (Luk II) in view of "Exploiting Hardware Performance Counters with Flow and Context Sensitive Profiling" by Ammons et al. (Ammons).

Regarding Claim 3: The rejection of claim 2 is incorporated; further Luk I discloses generating one or more profiles for an application (pg. 43, the par. bridging the cols. "decide how to prefetch … based on profiling information") but Luk I and II do not explicitly disclose executing the application with debug information or sampling cache misses and accumulating hardware counters for each static load of the selected code region.

Ammons teaches generating one or more profiles comprises:

executing an application associated with the main thread with debug information (pg. 86, col. 1, last full par. "a tool … that instruments program executables"); and

sampling cache misses and accumulating hardware counter for each static load of the code region to generate the one or more profiles for each cache hierarchy (pg. 85 col. 2 1st full par. "exploits the hardware performance counters").

It would have been obvious to one of ordinary skill in the art at the time the invention was made to implement Luk I's profiling (pg. 43, the par. bridging the cols. "based on profiling information") using Ammons methods (pg. 86, col. 1, last full par.; pg. 85 col. 2 1st full par.) Those of ordinary skill in the art would have been motivated to do so in

order to achieve the improved profiling disclosed (Ammons pg. 85, col. 2, 1st full par. "extends profiling techniques in two new directions").

Regarding Claim 4: The rejection of claim 3 is incorporated; further Luk I discloses analyzing the one or more profiles comprises:

correlating the one or more profiles with respective source code based on the debug information (pg. 41, col. 1, the last partial par. "decide where to launch pre-execution in the program, based on … cache miss profiling").

Luk I and II do not disclose identifying top loads that contribute to cache misses.

Ammons teaches identifying top loads that contribute cache misses above a predetermined level as the delinquent loads (pg. 86, col. 2, 1st partial par. "1% of the paths … account for 42 and 56% of the misses.").

It would have been obvious to one of ordinary skill in the art at the time the invention was made to identify the top loads discussed in Ammons (pg. 86, col.2, 1st partial par.) in Luk I's profile data (pg. 43, the par. bridging the cols. "based on profiling information"). Those of ordinary skill in the art would have been motivated to do so in order to balance the number of helper threads that are created with the effectiveness of each thread.

Claims 5-6, 17-19, 22-25 and 28-29 are rejected under 35 U.S.C. 103(a) as being unpatentable over "Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors" by Luk (Luk I) in view of US 2003/0084433 to Luk et al. (Luk II) in view of "Data Prefetching by Dependence Graph Precomputation" by Annavaram et al. (Annavaram).

Regarding Claim 5: The rejection of claim 1 is incorporated; further Luk I & II do not disclose building a dependent graph and performing slicing based on the dependent graph. Luk I does disclose "a collection of schemes … have been proposed to construct and pre-execute slices" (pg. 50, col. 1, the last partial par.)

Annavaram teaches building a dependent graph that captures data and control dependencies of the main thread (pg. 1 Abstract "efficiently generates the required dependence graphs"); and

performing a slicing operation on the main thread based on the dependent graph to generate code slices, each code slice corresponding to one of the one or more delinquent loads (pg. 1 Abstract "generate the data addresses of the marked load/store instructions"; pg. 9, par. bridging col. 1 & 2 "slices are preexecuted … for early generation of load addresses").

It would have been obvious to one of ordinary skill in the art at the time the invention was made to integrate Anavaram's dependent graph and associated slicing operation

with Luk I's system. Those of ordinary skill in the art would have been motivated to do so because Luk I discloses "[His] approach and [Anavaram's] can be complementary" (see Luk I pg. 50, col. 2, 1st partial par.)

Regarding Claim 6: The rejection of claim 5 is incorporated; further Luk I discloses selecting the code region further comprises:

computing liveness information providing communication cost between the main thread on the one of the helper threads (pg. 49, col. 2, 1st full par. "decide which address being accessed in pre-execution (and their surrounding control flow) are mostly communicated through registers.");

determining a communication scheme communicating live-in values between the main thread and the helper thread (pg. 44, Fig. 4 "Proposed instruction set extensions to support pre-execution") according to the liveness information, wherein the live-in values are accessed in the helper thread without re-computation (pg. 49, col. 2, 1st full par. "decide which address being accessed in pre-execution (and their surrounding control flow) are mostly communicated through registers.").

Annavaram teaches limiting traversal of the dependency graph to be within the code region for the slicing operations (pg. 4, 2nd full par. "follows only the predicted control flow path from the branch predictor");

merging two or more of the code slices into a helper thread of the one or more helper threads to minimize code duplication (pg. 9, col. 2, 1st partial par. "a chaining

trigger mechanism whereby speculative threads are also allowed to spawn other speculative threads");

determining a change in size of the helper thread according to one of the slicing operations corresponding to a separate code region of the one or more overlapping code regions, wherein the code region encompasses the separate code region, wherein the change reduces the size of the helper thread (pg. 9, col. 1, 2nd full par. "uses hardware to generate its dependence graphs dynamically, which significantly reduces the graph size and makes precomputation quite feasible").

It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the references as discussed in the parent claim.

Regarding Claim 17: Luk I discloses a method, comprising:

executing a main thread of an application in a multi-threading system (pg. 40, col. 1, 2nd par. "single threads running on multithreaded processor"); and

spawning one or more helper threads from the main thread having source codes including one or more code regions sharing at least one instruction in the source codes (e.g. Fig. 1(d)), the code region corresponding to a sequence of instructions representing an iteration loop in the source codes, the one or more helper thread to perform one or more computations for the main thread when the main thread enters a code region selected from the one or more code regions, the selected code region having one or more delinquent loads (pg. 40, col. 1, 2nd par. "spawning helper threads

… generates data addresses, on behalf of the main thread"), during a compilation of the main thread (pg. 44, col. 2, 3rd full par. "the compiler … is responsible for inserting pre-execution … performs all necessary code transformations"), wherein the one or more helper threads are synchronized with the main thread based on a synchronization period determined for the one or more helper threads, so that the one or more helper threads perform the one or more computations during the synchronization period (pg. 42, col. 1, 3rd full par. "PreExecute_Stop(), which will terminate the pre-execution and free up T for future use").

Luk I does not explicitly disclose each code region associated with an estimation of communication cost with the main thread and generating one or more helper threads based on the estimation.

Luk II teaches code regions associated with an estimation of communication cost with the main thread and generating one or more helper threads based on the estimation (par. [0029] "examining the benefit of load prefetching versus the cost in including the prefetches … The costs generally include the overhead associated with prefetch instructions the additional memory bandwidth consumed").

It would have been obvious to one of ordinary skill in the art at the time the invention was made to estimate communication costs between the main thread and each code region as taught by Luk II (par. [0029] "examining the benefit of load prefetching versus

the cost in including the prefetches") when selecting Luk I's code regions (pg. 44, col. 2 3rd full par. "locality analysis phase which determines which references … could benefit from pre-execution"). Those of ordinary skill in the art would have been motivated to do so to ensure that prefetching would actually provide a benefit (Luk I par. Luk II par. [0030] "favors prefetching a load if the data can be prefetched and then loaded in fewer clock cycles than it would take to load the data from memory").

Luk I & II do not disclose code of the one or more helper threads begin created separately from the source codes of the main thread.

Annavaram teaches one or more helper threads being created separately from source codes of the main thread (Abstract "generates the required dependence graphs at runtime … executes these graphs to generate the data addresses").

It would have been obvious to one of ordinary skill in the art at the time the invention was made to integrate Anavaram's dependent graph and associated slicing operation with Luk I's system. Those of ordinary skill in the art would have been motivated to do so because Luk I discloses "researchers have investigated ways to pre-execute only a subset of instructions ... our approach and [Anavaram's] can be complementary" (see Luk I pg. 50, bridging the cols.).

Regarding Claim 18: The rejection of claim 17 is incorporated; further Luk I discloses:

creating a thread pool to maintain a list of thread contexts (pg. 42, col. 1, 1st full par. "N hardware contexts supported by the machine"); and

allocating one or more thread contexts from the thread pool to generate the one or more helper threads (pg. 41, col. 1, the last partial par. "Each thread-spawning instruction requests for an idle hardware context to pre-execute the code sequence").

Regarding Claim 19: The rejection of claim 18 is incorporated; further Luk I discloses:

terminating the one or more helper threads when the main thread exits the code region (pg. 46, col. 1, 3rd par. "terminate a pre-execution thread if … the main thread has executed N instructions after passing P"); and

releasing the thread contexts associated with the one or more helper threads back to the thread pool (pg. 41, col. 2, the 1st partial par. "T will free its hardware context").

Regarding Claim 22: The rejection of claim 17 is incorporated; further Luk I discloses discarding results generated by the one or more helper threads when the main thread exits the code region, the results not being reused by another code region of the main thread (pg. 41, col. 2, the 1st partial par. "results held in T's registers are simply discarded").

Regarding Claims 23-25 and 28: Claims 23-25 and 28 recite a computer readable storage medium for instructing a computer to perform the methods of claims 17-19 and 22 and are addressed similarly.

Regarding Claim 29: Claim 29 recites a system for performing the method of claim 17 and is addressed similarly.

Claims 8-9 are rejected under 35 U.S.C. 103(a) as being unpatentable over "Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors" by Luk (Luk I) in view of "A Study of Source-Level Compiler Algorithms for Automatic Construction of Pre-Execution Code" by Kim and Yeung (Kim).

Regarding Claim 8: Luk I discloses a machine-readable storage medium having executable code to cause a machine to perform a method, the method comprising:

analyzing source codes of a main thread having one or more delinquent loads, the one or more delinquent loads representing loads which likely suffer cache misses during an execution of the main thread (pg. 44, col. 2 3rd full par. "locality analysis phase which determines which references are likely to cause cache misses"; also see Appendix Phase I, Step 1; pg. 45, "data address generation and the surrounding control flow"), the source codes including one or more code regions, each code region corresponding to a sequence of instructions representing an iteration loop in the source

codes, the one or more code regions sharing at least one instruction in the source codes (pg. 42, col. 1, 2nd full par. "when we are still working on the current list … nodes on the next list can be loaded speculatively by pre-executing the next iteration of the for-loop"; pg. 42 Fig. 2(b), "PreExecute_Start(END_FOR); … PreExecute_Stop()" and the corresponding region in Fig. 2(a));

selecting a code region from the one or more code regions for one or more helper threads with respect to the main thread based on the analysis (pg. 44, col. 2 3rd full par. "locality analysis phase which determines which references … could benefit from pre-execution"; also see the Appendix Phase I, Step 2 "control-flow and call-graph analysis"); and

generating software codes for the one or more helper threads, the one or more helper threads to be speculatively executed in parallel with the main thread to perform one or more prefetching tasks for the selected code region of the main thread (pg. 44, col. 2, 3rd full par. "performs all necessary code transformations"; also see the Appendix Phase II),

wherein the generated software codes include synchronization codes for the one or more helper threads to synchronize with the main thread during the execution (pg. 42, col. 1, 3rd full par. "PreExecute_Stop(), which will terminate the pre-execution and free up T for future use"; note that the claim does not put any limitations on the type of synchronization code, thus this code which returns execution to the main thread meets the broadly claimed limitation).

Luk I does not disclose wherein synchronization codes are generated based on a synchronization period for the one or more helper threads to synchronize with each other during the execution, and wherein the synchronization period is based on a distance between the one or more helper threads and the main thread.

Kim teaches synchronization codes are generated based on a synchronization period for the one or more helper threads to synchronize with each other during the execution (pg. 34, 1st full par. “our compiler uses counting semaphores to synchronize pre-execution threads with the main thread, and ... also to synchronize between pre-execution threads"), and wherein the synchronization period is based on a distance between the one or more helper threads and the main thread (pg. 24, 2nd par. “the main thread … incrementing the [global] counter … the pre-execution thread … increments its private counter, and compares the count to the global counter. The pre-execution thread continues only if the difference between the counters does not exceed PD, the prefetch distance”).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to implement Luk I's pre-fetching (pg. 42, col. 1, 2nd full par. “when we are still working on the current list … nodes on the next list can be loaded speculatively by pre-executing the next iteration of the for-loop”) using Kim’s thread synchronization controls (pg. 34, 1st full par. “our compiler uses counting semaphores to synchronize pre-execution threads with the main thread”). Those of ordinary skill in the art would

have been motivated to do so as an alternate method of performing the synchronization which would have produced only the expected results (Luk I, pg. 44, col. 2, 1st partial par. "the ISA can be further extended to have different favors of control over pre-execution") and would keep the pre-execution threads from running too far ahead (Kim pg. 20, last par. "We insert a single semaphore, called "T_0," to keep the backbone thread from getting more than "PD" iterations ahead of the main thread").

Regarding Claim 9: The rejection of claim 8 is incorporated; further Luk I discloses analyzing the source codes comprises:

generating one or more profiles for cache misses of the selected code region (pg. 43, the par. bridging the cols. "based on profiling information"; also see the Appendix, Phase I, Step 1 "This step can be accomplished through some low-overhead profiling tools"); and

analyzing the one or more profiles to identify one or more candidates for thread-based prefetch operations (pg. 43, the par. bridging the cols. "the compiler usually needs to heuristically decide how to prefetch … based on profiling information").

Claims 10-11 are rejected under 35 U.S.C. 103(a) as being unpatentable over "Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors" by Luk (Luk I) in view of "A Study of Source-Level Compiler Algorithms for Automatic Construction of Pre-Execution

Code" by Kim and Yeung (Kim) in view of "Exploiting Hardware Performance Counters with Flow and Context Sensitive Profiling" by Ammons et al. (Ammons).

Regarding Claims 10-11: Claims 10-11 recite a computer readable medium for instructing a computer to perform the methods of claims 3-4 and are addressed similarly.

Claims 12-14 are rejected under 35 U.S.C. 103(a) as being unpatentable over "Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors" by Luk (Luk I) in view of "A Study of Source-Level Compiler Algorithms for Automatic Construction of Pre-Execution Code" by Kim and Yeung (Kim) in view of "Data Prefetching by Dependence Graph Precomputation" by Annavaram et al. (Annavaram).

Regarding Claims 12-14: Claims 12-14 recite a computer readable medium for instructing a computer to perform the methods of claims 5-6 and are addressed similarly.

Claims 20-21 and 26-27 are rejected under 35 U.S.C. 103(a) as being unpatentable over "Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors" by Luk (Luk I) in view of US 2003/0084433 to Luk et al. (Luk II) in view of "Data Prefetching by Dependence

Graph Precomputation" by Annavaram et al. (Annavaram) in view of US 7,243,267 to Klemm et al. (Klemm).

Regarding Claim 20: The rejection of claim 17 is incorporated; further Luk I discloses wherein the one or more help threads are placed in a run queue prior to execution (pg. 46, col. 1, last full par. "instruction queue"), further comprising:

determining a period for each of the helper threads in the run queue, each of the helper threads being terminated from the run queue when the respective period expires, thereby releasing resources back to the main thread (pg. 46, col. 1, 2nd par. "Once this limit is reached, the thread will be terminated anyway").

The Luk I-Luk II-Annavaram combination does not explicitly disclose the period is a time period.

Klemm teaches determining a time period for a thread (col. 5, lines 57-58 "thread execution time exceeds user-specified threshold").

It would have been obvious to one of ordinary skill in the art at the time the invention was made to terminate one of Luk I's helper threads after a time period expires (Klemm col. 5, lines 57-58 "thread execution time exceeds user-specified threshold") as an alternate or additional instance of Luk I's "system-enforced terminating conditions for preserving correctness or avoiding wasteful computation" (col. 46, col. 1, 1st par.)

Regarding Claim 21: The rejection of claim 20 is incorporated; further Luk I discloses each of the helper threads terminates when the period expires even if the respective helper thread has not been accessed by the main thread (pg. 46, col. 1, 2nd par. "the thread will be terminated anyway").

Regarding Claims 26-27: Claims 26-27 recite a computer readable medium for instructing a computer to perform the methods of claims 20-21 and are addressed similarly.

Conclusion

In view of the new grounds of rejection **this action is made NON-FINAL.**

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Jason D. Mitchell whose telephone number is (571)272-3728. The examiner can normally be reached on Monday-Thursday and alternate Fridays 7:30-5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Bullock Lewis can be reached on (571) 272-3759. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Jason D. Mitchell/
Primary Examiner, Art Unit 2193